mathematics

MDPI

Article

Mining Significant Utility Discriminative Patterns in Quantitative Databases

Huijun Tang [1,2], Jufeng Wang [1,*] and Le Wang [3]

[1] Faculty of Finance and Information, Ningbo University of Finance & Economics, Ningbo 315175, China
[2] Faculty of Electrical Engineering and Computer Science, Ningbo University, Ningbo 315211, China
[3] Faculty of Digital Technology and Engineering, Ningbo University of Finance & Economics, Ningbo 315175, China
* Correspondence: wjf@nbt.edu.cn

Abstract: Drawing a discriminative pattern in quantitative datasets is often represented to return a high utility pattern (HUP). The traditional methods output patterns with a utility above a pre-given threshold. Nevertheless, the current user-centered algorithm requires outputting the results in a timely manner to strengthen the interaction between the mining system and users. Pattern sampling can return results with a probability guarantee in a short time, and it could be a candidate technology to mine such discriminative patterns. In this paper, a novel approach named HUPSampler is proposed to sample one potential HUP, which is extracted with probability significance according to its utility in the database. HUPSampler introduces an interval constraint on the length of HUP and randomly extracts an integer k according to the utility proportion firstly; then, the HUPs could be obtained efficiently from a random tree by using a pattern growth way, and finally, it returns a HUP of length k randomly. The experimental study shows that HUPSampler is efficient in regard to memory usage, runtime, and utility distribution. In addition, case studies show that HUPSampler can be significantly used in analyzing the COVID-19 epidemic by identifying critical locations.

Keywords: high utility pattern; sampling; quantitative database; COVID-19

MSC: 68-04

Citation: Tang, H.; Wang, J.; Wang, L. Mining Significant Utility Discriminative Patterns in Quantitative Databases. *Mathematics* **2023**, *11*, 950. https://doi.org/10.3390/math11040950

Academic Editor: Yumin Cheng

Received: 16 December 2022
Revised: 25 January 2023
Accepted: 8 February 2023
Published: 13 February 2023

1. Introduction

Pattern Mining is one of the key technologies of big data analysis [1–3]. HUP mining is used to discover patterns in quantitative databases whose utilities are bigger than a pre-given utility threshold. HUP has utility discriminative characteristics, and many algorithms have been proposed for mining kinds of HUP types in quantitative databases, including traditional high utility itemsets mining [4], frequent high utility itemsets mining [5], cross-level high utility itemsets mining [6], etc.

Different from traditional frequent pattern mining, mining HUP seems more complex because HUP mining focuses on both internal and external utility factors of itemsets rather than their frequency. In addition, downward closure property [4], which is known as an effective strategy in frequent pattern mining for reducing the mining computations, cannot be referenced in the HUP mining process. In order to achieve the goal of mining HUPs more efficiently, some algorithms have been introduced to process such problem, including ULBMiner [7], FHM [8], D2HUP [9], EFIM [10], HUI-Miner [11], FHM+ [12], etc. However, it is important to note that all these algorithms return the same HUPs under a pre-given utility parameter. Therefore, the difference between such algorithms is not their output, but how each algorithm finds the results; most of the algorithms focused on mining improvement of time-space efficiency.

It will take huge computations to mine HUPs on large datasets. A feasible method is to extract only a sample of the data since the computation of mining algorithms based

on the sample could be greatly reduced. Several size-sampling methods have been proposed. Riondato and Upfal present a progressive sampling method under the control of VC dimension [13] or Rademacher averages [14] to get the approximate frequent items, Djenouri et al. [15] use the heuristics method to obtain the sampling size. Parthasarathy [16] proposes progressive sampling-based algorithms based on frequency constraints, which determine the stopping size. Many effective data structures (such as pattern trees [17], link lists [18], header tables [19], etc.) are also used to search the approximate results based on sampling theory. The biggest challenge for sampling-based methods is how to ensure the effectiveness of the results obtained on the sample set. Some useful methods are utilized to settle such problems. Interval estimation is used to mine approximate association rules [20] or frequent itemsets [21] based on sample theory. Bashir [22] uses the strategy of inequality control to guarantee the results for frequent itemsets mining. Yan Chen [23] first introduces Hoeffding's inequality and Chebyshev's inequality [24] to mine approximate HUPs, but its probability-bound guarantee may be relatively loose.

These sampling methods, which usually return an effective sample size, can be summarized as firstly reasoning an integer size with sample theory, and then mining-related HUPs based on this size by probability guarantees to ensure the significance of sample-based mining results. The progressive sampling method does not fix the sample size and it relies on bounding the stopping condition of the sample. The computational complexity of progressive sampling may be related to the characteristic of the dataset (dense or sparse), which cannot completely ensure outputting patterns timely. More importantly, recently, user-centered pattern mining algorithms have become a new research trend [25–27], requiring that the generated patterns are not only timely but also diversified. All the complete mining methods, which return exact HUPs cannot finish the task of abstracting HUP in a short response time. Fixed size-based sampling methods may still be time-consuming, as they only extract a size with accuracy bound on the whole dataset. For example, on the chess dataset [28], such a method takes more than 10^6 s under a 90% probability guarantee and more than 10^4 s under a 50% probability guarantee; the size-based method cannot meet our goal of real-time interaction between the patterns and the users. Recently, some advanced sampling algorithms are not going to return the pattern set that is based on the user's requirements, and they just return one pattern that is discriminative with high probability. Algorithm CSSAMPLING [29] extracts sequential patterns randomly based on the length constraints of the patterns. The sampling patterns returned by such an approach are according to the probabilities of frequency. Firstly, according to the proportion of frequency, draw one transaction randomly, and then return a pattern randomly which is proportional to its frequency in the transaction extracted before. HAISampler [30] considers the problem of the pattern drawing with utility. It outputs the patterns, which are proportional to their average utility in the transaction. Unfortunately, it can indeed return results in a relatively short time, but the returned results may not be HUP, which provides research inspiration for us to pursue returning one HUP in a short time.

In this study, we propose HUPSampler, which returns a HUP randomly according to the utility proportion and can meet the requirements of users, returning the results in a short time and strengthening the interaction between the users and the mining system. Meanwhile, it can present a set of varied patterns in a short time so as to have a tight interaction between the two parts. Such a method has hitherto been considered, and it relies on a two-phase procedure.

In Step 1, two positive integers m and M ($m < M$) are given by the users; we draw a length k according to the utility proportion of the patterns of length k. Firstly, for each length in the interval of $[m, M]$, we can obtain the total utility of patterns of each length by scanning the dataset once based on an efficient tree structure, and then an integer k is drawn randomly according to the utility probability of patterns with length k.

In Step 2, we randomly draw a HUP of length k which is obtained by Step 1. Firstly, the HUP set of length k is efficiently mined from the tree established in Step 1. This number

k may be a most probable length in the interval of $[m, M]$ according to the utility proportion. Then, we output one HUP of length k uniformly so as to reflect diversity in each interaction.

The rest of this paper is as follows. Section 2 defines relevant terms. Section 3 gives our corresponding algorithm HUPSampler. Section 4 shows experimental results, and Section 5 gives conclusions.

2. Preliminaries and Problem Statement

A transactional dataset D = {T_1, T_2, T_3, . . . , T_n}, T_d is one transaction with unique identification d. There are n unique items I = {i_1, i_2, . . . , i_n}. T_d may contain several items in I and its corresponding number, which is donated as q (i_j, T_d), e.g., T_1 = {(B, 4) (C, 3) (B, 3)} in Table 1. An item i_j also has a profit p (i_j), e.g., p (A) = 2 in Table 2. The notations of this paper are summarized in Table 3.

Table 1. An example of a transactional dataset.

	Transaction
T_1	(B, 4) (C, 3) (E, 3)
T_2	(A, 1) (C, 4) (F, 10)
T_3	(A, 4) (B, 3) (E, 2) (F, 6)
T_4	(B, 1) (D, 1)
T_5	(A, 3) (B, 3) (C, 2) (D, 1) (E, 2)
T_6	(A, 3) (C, 2) (D, 1)

Table 2. Profits.

Item	**A**	**B**	**C**	**D**	**E**	**F**
Profit	2	5	4	6	10	1

Table 3. Notations.

Symbol	**Definition**
HUP	High utility pattern
HUPK	High utility pattern of length k
D	Transactional Dataset
T_d	dth transaction in D
$HUP_{[m, M]}$	Set of patterns with length in $[m, M]$
$MHUPK$	Algorithm of mining HUPK
$TU_{[m, M]}$	Sum list of transaction utility with length in $[m, M]$
TWU	Transaction-weighted utility value of D
$RTWUK$	The maximum utility of patterns of length k in D
TU_k	Sum of transaction utility of length k
$U(X)$	Utility of pattern X
$U(T_d)$	Utility of transaction T_d
δ	Minimum utility threshold
ID-tree	A utility tree
m	Minimum lengthof HUP
M	Maximum length of HUP

Definition 1. *$U(i_j, T_d)$ is defined as utility value of i_j in T_d, and*

$$U(i_j, T_d) = \mathrm{p}(i_j) \times \mathrm{q}(i_j, T_d) \quad (1)$$

In Tables 1 and 2, $U(B, T_1) = 5 \times 4 = 20$, $U(C, T_1) = 3 \times 4 = 12$, and $U(E, T_1) = 3 \times 10 = 30$.

Definition 2. *$U(X, T_d)$ is defined as utility value of X in T_d, and*

$$U(X, T_d) = \begin{cases} \sum_{i_j \in X} U(i_j, T_d), & if X \subseteq T_d \\ 0 & else \end{cases} \quad (2)$$

In Tables 1 and 2, U ({BC}, T_1) = 32, U({BE}, T_1) = 50.

Definition 3. *U (X) is defined as the utility of itemset X in D, and*

$$U(X) = \sum_{T_d \in D \wedge X \subseteq T_d} U(X, T_d) \quad (3)$$

In Tables 1 and 2, U ({CE}) = U ({CE}, T_1) + U ({CE}, T_5) = 42 + 28 = 70.

Definition 4. *TU (T_d) is defined as the utility value of transaction T_d, and*

$$TU(T_d) = \sum_{i_j \in T_d} U(i_j, T_d) \quad (4)$$

In Tables 1 and 2, TU (T_4) = U (B, T_4) + U (D, T_4) = 5 + 6 = 11.

Definition 5. *TU is defined as utility value of D, and*

$$TU = \sum_{T_d \in D} TU(T_d) \quad (5)$$

In Tables 1 and 2, TU = 62 + 28 + 49 + 11 + 55 + 20 = 225.

Definition 6. *TWU(X) is defined as transactional weighted utility of X, and*

$$TWU(X) = \sum_{T_d \in D \wedge T_d \supseteq X} TU(T_d) \quad (6)$$

Definition 7. *MinU is defined as the minimum utility value based on a threshold δ, which is known as a user-specified percentile of total transaction utility values of the given dataset D, and*

$$MinU = TU \times \delta \quad (7)$$

Definition 8. *X is a HUP if U(X) ≥ MinU. A HUP of length k is denoted as HUPK.*

Property 1. *For a pattern X, if TWU(X) < MinU, X will not be a HUP.*

Proof. *TWU (X) ≤ MinU*. That is to say, the sum of transaction utility of *X* is less than *MinU*, *U (X) ≤ TWU (X)*, *X* will not be a candidate. □

Definition 9. *TUK (T_w, k) is defined as the maximum sum of utility of k items in T_w, and*

$$TUK(T_w, k) = \begin{cases} \max(\sum_{j=1}^{k} U(x_{i_j}, T_d) \Big| x_{i_j} \in T_w) & if \ |T_w| \geq k \\ 0, & else \end{cases} \quad (8)$$

In Tables 1 and 2, TUK (T_1,3) = max (U (BCE, T_1,)) = max (62) = 62.

Definition 10. *TUK (X, T_w, k) is defined as maximum utility in T_w of length k containing X, and*

$$TUK(X, T_w, k) = \begin{cases} U(X, T_w) + TUK(T_w - X, k - |X|) & if \ X \subseteq T_w \wedge |T_w| \geq k \\ 0, & else \end{cases} \quad (9)$$

In Tables 1 and 2, TUK ({C}, T_1, 2) = U (C, T_1) + TUK({T_1 − {C}, 1) = 12 + max (U (B, T_1), U (E, T_1)) = 12 + max (20, 30) = 42.

Definition 11. *RTWUK (X, k) is defined as the maximum utility of length k containing X in D, and*

$$RTWUK(X,k) = \sum_{T_w \in D \wedge T_w \supseteq X} TUK(X, T_w, k) \tag{10}$$

In Tables 1 and 2, RTWUK ({E}, 2) = TUK ({E}, T_1, 2) + TUK ({E}, T_3, 2) + TUK ({C}, T_5, 2) = 50 + 35 + 35 = 120.

Definition 12. *X is considered as a candidate of HUPK if its RTWUK < MinU.*

The detailed instructions of Definitions 9–12 can be found in Ref. [31], and we use such a RTWUK property to draw a HUP randomly.

We aim to randomly extract a HUP according to the utility proportion. Based on the pre-given two parameters m and M, $HUP_{[m,M]}$ is donated as the set of all patterns with length in the interval $[m, M]$. We draw a HUP S with a significance-based utility discriminative value $P(S)$ in D, i.e.,

$$P(S) = \frac{U(S)}{\sum_{S' \in HUP_{[m,M]}} U(S')} \tag{11}$$

$U(B) = 22$, $U(C) = 44$, if $m = 1$, $M = 1$, our objective is to develop a sampling method such that the probability to draw C is two times more than the that of extracting B.

3. Two-Phase Sampling Procedure

3.1. Overview of the Procedure

We are going to sample a HUP proportionally to its utility in the length internal of $[m, M]$ based on a two-phase procedure in [29,30]. HUPSampler returns a high utility pattern of length between m and M. The whole procedure of the HUPSampler could be seen in Algorithm 1.

Algorithm 1: HUPSampler

Input: D: Dataset
$[m, M]$: the user-specified length interval
δ: minimum utility threshold
Output: a random HUP
//Step 1: Sampling an integer k
1. Create an ID-tree by scanning D once.
2. For each integer j in $[m, M]$, calculate TU_j, $TU_{[m,M]}$ and TU based on ID-tree
3. Draw an integer k in $[m, M]$ based on the utility distribution $TU_k \sim TU_{[m,M]}$
//Step 2: Sampling a HUP
4. Create an ID-tree of RTWUK with random item sequence
5. Randomly return a HUP of length k with utility $> \delta \cdot TU$ based on ID-tree

For calculating the utility of TU_j ($j = m, m + 1, \ldots, M$), by scanning D, we first establish a novel tree structure ID-Tree (Line 1), which stores compressed original data as well as pattern indexing information. The same items of the transactional data are compressed to the same node. When each transaction is added to the ID-tree, leaf-nodes record the utility of the candidates, TU_j is calculated by a pattern growth method, $TU_{[m,M]}$ could be calculated cumulatively (Line 2), and we can draw integer k randomly in accordance with the probability of $TU_j/TU_{[m,M]}$ (Line 3). In the second phase, we randomly output a HUP of length k from D. Firstly, a tree with a random item sequence by the RTWUK proportion of utility is established (Line 4). Then, we draw one HUP of length k randomly based on the new tree, and once a HUP is satisfied, we output such a HUP in a timely manner, and the pursuit procedure is done (Line 5).

3.2. ID-Tree for Drawing Integer k

3.2.1. Structure of ID-Tree

For storing the sequence information in an ID-tree, each node N_d contains three ordinary fields:

- N_d.name records the name of the item in N_d;
- N_d.parent records the parent of N_d;
- N_d.children records the children of N_d.

The leaf nodes are also known as tail nodes, and to each tail node in the tree, there exists a path from the tail node to the root node. The items in the path could be known as an itemset, and it contains the following fields:

- N_d.piu records the utility of the items in the path which is from N_d to the root node. And it can be visited by the indexes of the nodes;
- N_d.bu saves the utility list of base-item which contains the utility of N_d and could be known as a sub set of the path from N_d to the root node.

All patterns with required lengths could be known in the ID-Tree. That is to say, with one scanning, it could be known that the path from nodes in leaf node to the root store HUPs of any length. Based on the data in Tables 1 and 2, Figure 1 shows the creation of the ID-tree. The right most node D is a tail node; 8, 6, 6 on the node means the utilities of items C, A, D, respectively. They are sorted by the descending TWU of the items.

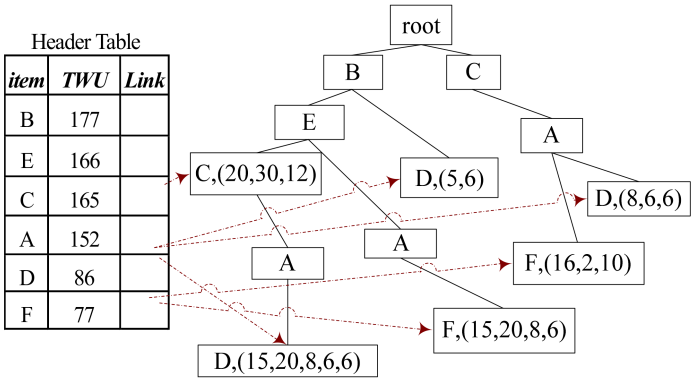


Figure 1. ID-tree constructed from the data in Tables 1 and 2.

3.2.2. ID-Tree Construction

An ID-Tree is established by scanning the dataset once. The pseudocode can be seen in Algorithm 2. By scanning transactions in D, we can calculate the *TWU* of the items in header table *H*, and they can be sorted by descending *TWU*. For an empty ID-tree, the items in each transaction, which are sorted as the items in *H*, can be added to the tree, and the tail node to the root consists of a path. The utilities of items in the path could be known in the list of the tail node.

We take the data in Tables 1 and 2 as an example to demonstrate the tree creation process which could be known in Figure 2. The header table in Figure 2a consists of two parts, which are TWU and the link pointer. With one scan, we create the header table *H*, sort the items in *H* by descending order, add T_1 to the tree in the same descending order, and save the utility of path in leaf node such as 'C' (20, 30, 12); in Figure 2b, 'C' is the leaf node, and the list shows the utility of items in this path. For the items in the header table, the link pointer is designed to draw a pointer, which is linked to the tail nodes of the tree, and by link point, the nodes can be visited conveniently. Adding each transaction to the tree sequentially, the tree after processing the second transaction T_2 can be seen in Figure 2c,d indicates the result after adding the third transaction. In the process of addition, if the transaction path already exists in the tree, you only need to add the corresponding utility. The result after adding all transactions to the tree is shown in Figure 2e.

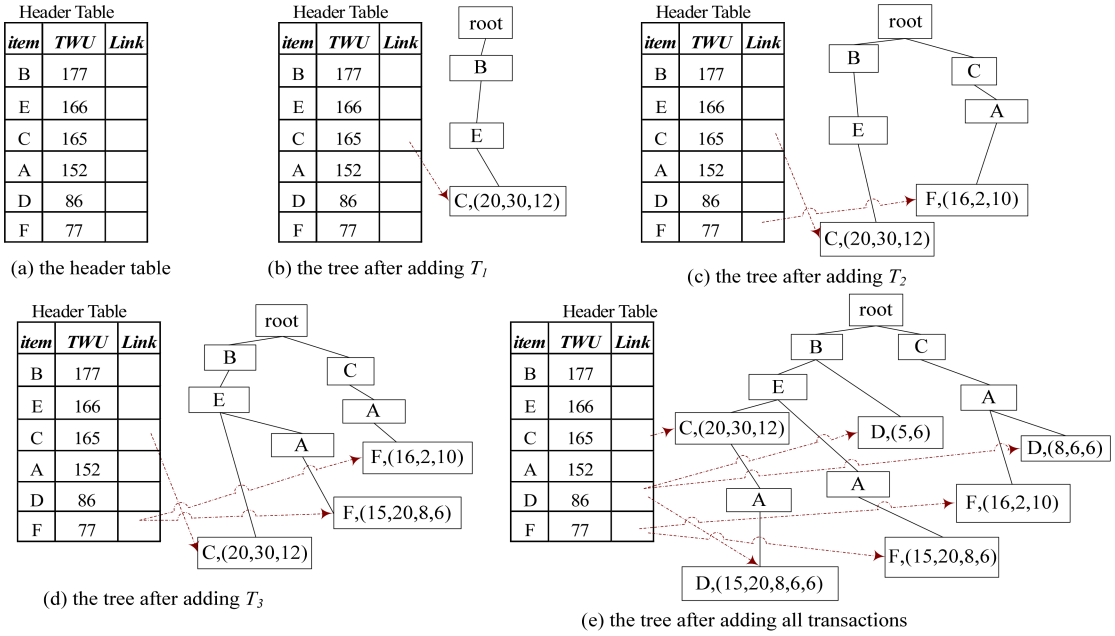


Figure 2. A case of creating a tree and a header table from the data in Tables 1 and 2.

3.2.3. Calculating $TU_{[m, M]}$ and Drawing Integer k Proportionally

The specific process of $TU_{[m, M]}$ calculation is shown as CalculateTU in Algorithm 3. Based on the ID-tree, we are going to calculate TU_j ($j = m, m + 1, \ldots, M$) based on the tree returned by Algorithm 2. It starts from each tail node of the ID tree (Line 1). The combined itemset from the tail to the root whose length is in $[m, M]$ could be known in Path (P, m, M) (Line 2), and we use a dictionary DC to accumulate the utility with each length in $[m, M]$ (Lines 3–5), then pass the utility list to the parent node (Lines 7–14) and delete the current item from the tree (Line 15). For a new T and H, we recursive call CalculateTU until all items in the original H are processed (Line 18).

Algorithm 2: CreateTree

Input: D: Dataset
Output: a ID-tree T
1. For each transaction T_d of D do
2. For each item X in T_d do
3. Calculate $H.X.TWU$
4. End For
5. End For
6. Initialize a Tree T with an empty root node, initialize a Table H with TWU descending order
7. T = root()
8. For each transaction T_d of D do
9. Insert T_d to T with the same descending order of H
10. Add utility to the leaf node
11. Add the links
12. End For
13. Return T

Algorithm 3: CalculateTU

Input: T: a tree
H: a header table
m: Minimum length
M: Maximum length
Dictionary DC
Output: TUj (j = m, m + 1, . . . , M)
1. For each tail node P in H do:
2. base-itemset = Path (P, m, M)
3. If len (base-itemset) $\geq m$ and len (base-itemset) $\leq M$ then
4. DC [len (base-itemset)] + =U (base-itemset)
5. End If
6. End For
7. For each tail node P in H do:
8. If P.parent.bu = NULL then
9. P.parent.bu = P.bu
10. P.parent.piu = P.piu
11. else:
12. P.parent.bu + =P.bu
13. P.parent.piu + =P.piu
14. End If
15. Remove P from T
16. End For
17. Create a header table $subH$
18. CalculateTU (T, $subH$, m, M,DC)

For the data in Tables 1 and 2, as an example when m = 2 and M = 5, firstly processing the first bottom tail node 'F', there are two paths where 'F' is the leaf node, <B,E,A,F> and <C,A,F>, thus the number and utility of patterns with 'F' and length in (2,5) could be easily calculated. For example, to the path<B,E,A,F>, there are three patterns of length 2 ({BF}, {EF}, and {AF}, respectively), three patterns ({BEF}, {BAF}, and {EAF}) of length 3, and one pattern ({BEAF}) of length 4. Then, tail nodes 'D' and 'C' are processed based on the same procedure, and TU_j (j = 2, 3, 4, 5) can be accumulated continuously. After processing these three tail nodes, the ID-tree T is updated according to the transition of the tail node; the tail node will pass the utility to its parent if the parent exists. Based on Figure 2, the updated ID-tree can be seen in Figure 3. There are two lists in the rightmost item 'A'. As two leaf nodes pass the utility to 'A', the TWU of each item should be re-calculated. According to the new ID-tree, Algorithm 3 will be recursively called until no pattern that appears on the tree meets the length requirement.

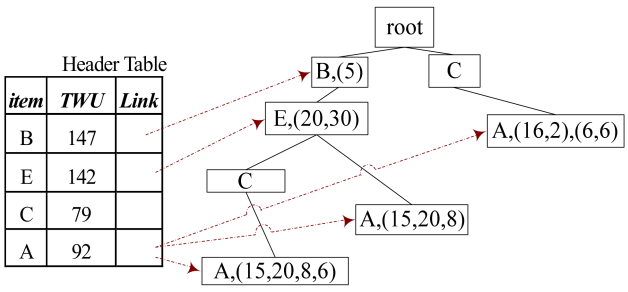


Figure 3. Update of ID-tree from the data in Tables 1 and 2.

According to the new tree in Figure 3, the final result of TU_j (j = 2, 3, 4, 5) could be known in Table 4. It is clearly known that the integer k = 2 or 3 may be selected with high probability due to its high utility.

Table 4. TU_j of data in Tables 1 and 2 when $m = 2$ and $M = 5$.

Length j	TU_j	$TU_{[2,j]}$
2	570	570
3	587	1157
4	269	1426
5	55	1481

3.2.4. Draw a HUP Uniformly

In this section, we will draw a HUP randomly based on the integer k extracted by Section 3.2.3. We propose an algorithm, called MHUPK (Mining HUPK) with RTWUK, based on Definitions 9–11.

MHUPK is going to pursue HUPKs, and it is also processed based on the tree structure. However, unlike the ID-tree, we look for HUPs of length k with RTWUK constraints, and because RTWUK < TWU, the computation of MHUPK could be reduced greatly. Additionally, we save RTWUK in the header table, which could be known as the updating of TWU. The process of MHUPK can be identified as Algorithm 4. The biggest difference is that we use RTWUK in the header table, and the items in the new header table are sorted randomly by the proportion of RTWUK.

If $MinU = 30$ and we are randomly drawing $k = 3$ from Table 4, taking the data in Tables 1 and 2 and based on the tree nodes in Figure 1, the RTWUK value of each item could be known in Table 5.

Table 5. RTWUK of the items when $k = 3$.

Item	A	B	C	D	E	F
RTWUK	132	148	143	61	150	69

To output an HUP of more than 30 randomly, we are going to establish a new tree by sorting the item randomly according to the proportion of RTWUK from small to big. That is to say, since the RTWUK of item 'B' is the biggest, it is ranked last by a high probability. The random tree with RTWUK may be in Figure 4.

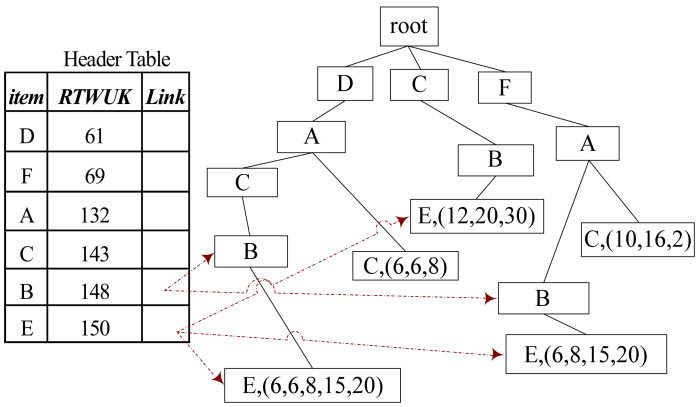


Figure 4. A case of creating a random tree and a random header table with RTWUK.

MHUPK is going to return one HUPK based on the random tree and header table in each iterative calculation. It is important to find the HUPs of certain base-items whose length are k, and the pattern whose length is less than k could be a candidate. A sub-tree and sub-table are established according to the pattern produced growth method. We visit the new sub table and sub tree recursively until one pattern is satisfied. The steps of MHUPK could be known in Algorithm 4.

Algorithm 4: MHUPK

Input: *H*: a random header table
T: a random tree with RTWUK corresponding to *H*.
k: length of pattern
MinU: minimum utility value
base-item
Output: A random HUP of length *k*
1. For each item P in H (with a bottom-up sequence) do:
2. If H.P.RTWUK $\geq$ *MinU* then
3. base-item = base-item $\cup${P}
4. If |base-item| = *k* and H.P.utility $\geq$ *MinU* then
5. Break and Return base-item
6. End If
7. If |base-item| < *k* then
8. Create SubHeader SubH and SubTree SubT
9. MHUPK (subT, subH, *k*, *MinU*, base-item)
10. End If
11. Remove P
12. End If
13. End For

Algorithm 4 is meant to mine one HUP from a tree with RTWUK. It consists of five steps: (1) For item *X* in the header table, if the RTWUK of *X* is less than *MinU*, such an item node should be removed from the tree, and we can begin to deal with the next item in the header table; otherwise, continue to carry out the next step. (2) Add the current processing item to the variable base-item, which is used to save the growth pattern, and the initialization of the base-item is null (Line 3). (3) When the base-item satisfies two conditions: one is that its length is *k*, and the other is that its utility is more than *MinU*, it could be discovered as a HUPK (Lines 4–6). (4) The sub header table and sub tree are created when then length of base-item is not more than *k*, and the algorithm will pursue HUPK based on the new header table and tree. This shows the idea of pattern growth, and one HUPK could be obtained by recursion (Lines 7–10). (5) When the current item is completed, it will be removed from the header table (Line 11), the next item in the original table will be processed until one HUPK is satisfied.

Figure 4 shows the corresponding header table and tree based on RTWUK, and we are going to output one HUPK from the tree. The item in the header table is randomly produced according to the RTWUK proportion, and item 'E' may be the last item in the table since it has the biggest RTWUK value. It can generate the items one by one, and the items in the random header table are sorted by the generation sequence of the items. Here, we assume that the generated header table is as shown in Figure 4, and we firstly process the last item 'E' in the header table. (1) It could know that the RTWUK of 'E' is 150, which is more than *MinU*. The item 'E' should be added to the set of the base-item, i.e., base-item = {E}. (2) According to the procedure of Algorithm 4, to the length of the base-item, which is less than 3, it will produce a corresponding random sub header table, which may be acted as Figure 5a. Thus, the corresponding sub-tree could be established by Figure 5b according to the new order. All the items could be added to the sub tree if its TWUK is more than *MinU*. (3) Continue to access the new table and tree. If the length of the new pattern has not reached 3, generate new random tables and the corresponding trees. In the mining process, once a pattern with a length of 3 and a utility meeting the requirements is found, it is outputted as HUPK, and the procedure is finished. Figure 5b is the sub tree of item 'E', and Figure 5c,d are the sub trees of itemset 'BE' and 'CE', respectively. Table 6 shows HUPs of length 3 based on the data in Tables 1 and 2, and we can draw one HUP with high probability such as {BCE} since it may be found earlier than other HUPs.

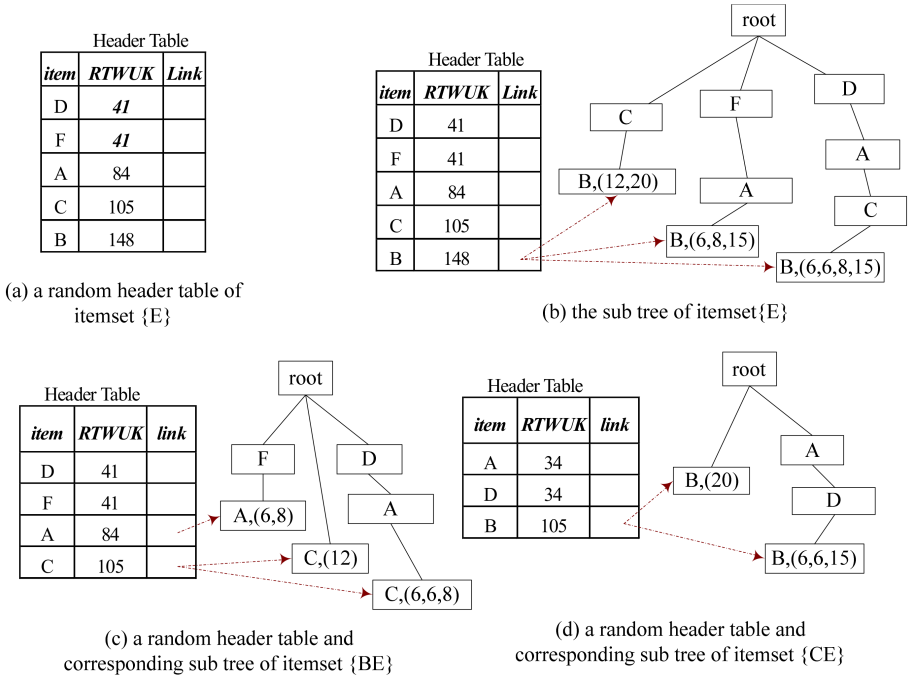


Figure 5. A case of creating a sub tree with RTWUK.

Table 6. HUPs of length 3 and $MinU \geq 30$.

Rank	*HUP*	*Utility*
1	{BCE}	105
2	{ABE}	84
3	{BEF}	41
4	{BDE}	41
5	{ACD}	40
6	{ACE}, {CDE}	34
7	{ADE}	32

4. Experimental Study

In this section, we evaluate the performance of the proposed algorithm HUPSampler on real datasets, including Chainstore, Pumsb, Retail, Connect, Mushroom, and Chess. All the datasets are from SPMF [28]. The utility setting is consistent with the existing HUP mining literature [32,33]. The details of these datasets are provided in Table 7, and the length constraint is initialized as (1,10). The experimental platform is configured as follows: Windows 10 operating system, 2G Memory, Intel (R) Core (TM) i3-2310 CPU@2.10 GHz.

Table 7. Dataset characteristics.

Dataset	Distinct Items (#)	Average Length	Transactions (#)
Chainstore	46,086	7.2	1,112,949
Pumsb	2111	74	49,046
Retail	16,470	10.3	88,162
Connect	129	43	67,557
Mushroom	119	23	8124
Chess	76	37	3196

4.1. Running Time

Table 8 shows the average running time and maximum memory consumption of 100 different runs on the six datasets under different minimum utility thresholds. HUPSampler can obtain the results in a short running time on each dataset. Since the numbers

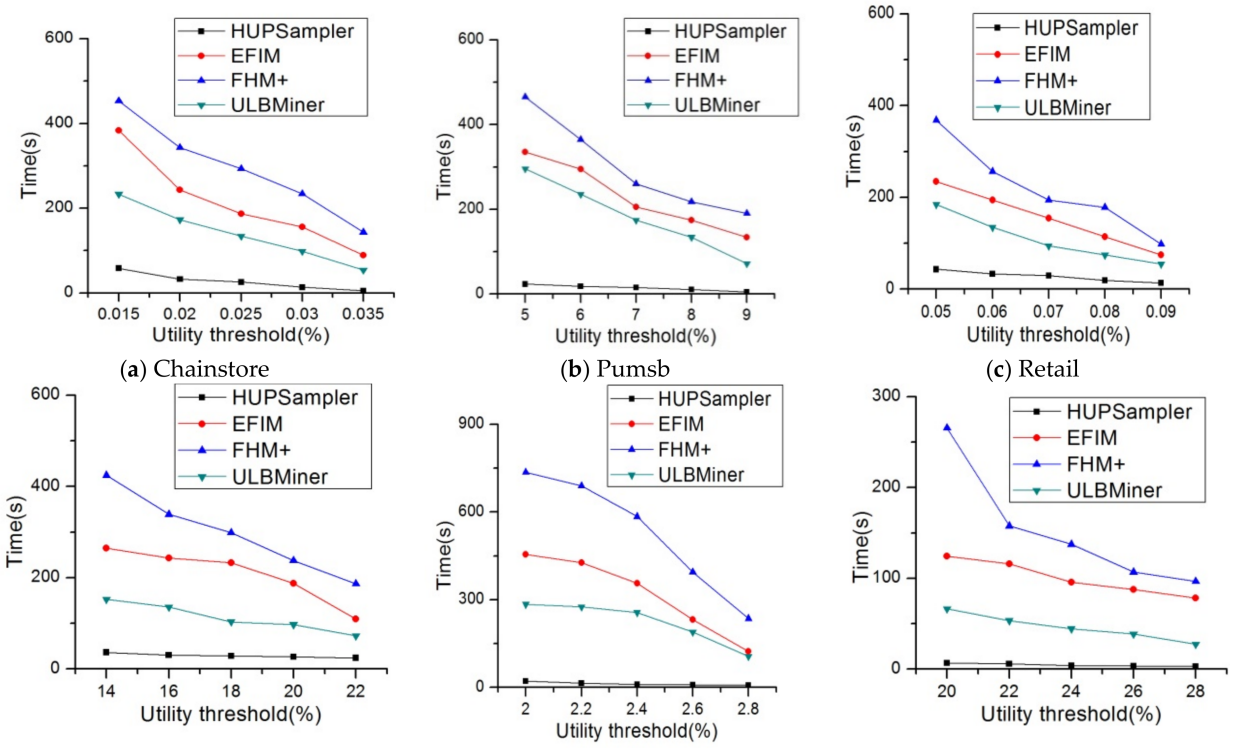
of HUPs generated under different thresholds are not the same, the smaller the threshold is, the more the number of HUPs will be generated. However, HUPSampler can always return patterns in a few seconds. For example, for each support threshold on Pumsb, Mushroom, and Chess, the running time of the algorithm HUPSampler is always less than 25 s. On sparse datasets such as Retail, the running time does not grow much; the sampling results can be obtained in less than 50 s. The HUPSampler only outputs one HUP, which may be with the largest utility proportion in the original transactional datasets, and the running time can be reduced greatly.

Table 8. Running time of HUPSampler under different utility thresholds.

Datasets	Minimum Utility Threshold (%)	Running Time (s)	Memory Usage (MB)
Chainstore	0.015	58.4	70.1
	0.02	32.6	53.8
	0.025	25.6	35.6
	0.03	13.5	21.8
	0.035	5.3	15.8
Pumsb	5	23.5	52.5
	6	18.5	30.7
	7	15.3	21.5
	8	10.5	11.5
	9	4.9	10.4
Retail	0.05	43.3	30.6
	0.06	33.5	21.6
	0.07	29.5	15.6
	0.08	19.6	11.7
	0.09	13.4	10.5
Connect	14	36.5	41.3
	16	30.6	40.6
	18	28.7	40.1
	20	26.5	38.7
	22	24.3	36.5
Mushroom	2.0	22.2	38.8
	2.2	14.5	36.9
	2.4	9.9	34.4
	2.6	8.4	30.4
	2.8	7.1	26.1
Chess	20	6.7	23.9
	22	5.9	22.2
	24	4.0	21.9
	26	3.6	13.8
	28	2.8	10.7

To further illustrate the effectiveness of the algorithm, HUPSampler is compared with three efficient HUP mining methods, which are EFIM [10], FHM+ [12], and ULBMiner [7], respectively. The running time results are shown in Figure 6. As we know that the running time of HUPSampler is significantly lower than the others on any minimum utility thresholds, we can clearly see that HUPSampler can draw HUP smoothly.

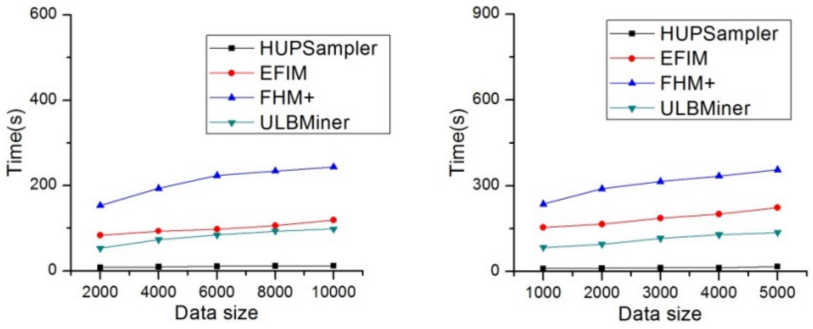
In view of the requirements for real-time returning of sampling results under different sizes, we also analyzed the running time of the algorithm based on different data volumes. As the amount of data increases, the running time of the algorithm increases by scanning the dataset. However, our results show that HUPSampler can always return results in a short time since it only outputs one HUP. Figure 7 shows the running time on datasets Chainstore and Mushroom compared with other algorithms under the utility thresholds of 0.04% and 2.5%, respectively. It is clearly seen that the performance of HUPSampler is stable. The results on other datasets can also achieve the same effect.



(**a**) Chainstore (**b**) Pumsb (**c**) Retail

(**d**) Connect (**e**) Mushroom (**f**) Chess

Figure 6. Running time comparisons of four algorithms.



(**a**) Chainstore (0.04%) (**b**) Mushroom (2.5%)

Figure 7. Running time under different data size.

4.2. Memory Usage

From Table 8, we can clearly see that the memory consumption of HUPSampler is always kept at a low level. Even with the constant reduction of the minimum utility parameter, it can always return a result in a memory less than 100 MB, which greatly improves the memory operation efficiency of the algorithm.

We also compared the memory consumption of HUPSampler with the three typical effective algorithms mentioned above. The relevant comparison results are shown in Figures 8 and 9, where Figure 8 returns the memory consumption under different minimum utilities on six data sets. Figure 9 shows the running results under different data volumes on ChainStore and Mushroom under the corresponding fixed minimum utility parameters. It can be significantly shown that from both figures, the memory consumption of HUPSampler has been greatly improved. For example, in Figure 8, when the minimum utility parameter is set to 0.07% on the Retail, the memory consumption of the other three algorithms is greater than 100 MB, while the memory usage of HUPSampler is only 15.6 MB. The same conclusion can also be shown on the Chainstore and Mushroom in Figure 9 based on different data sizes.

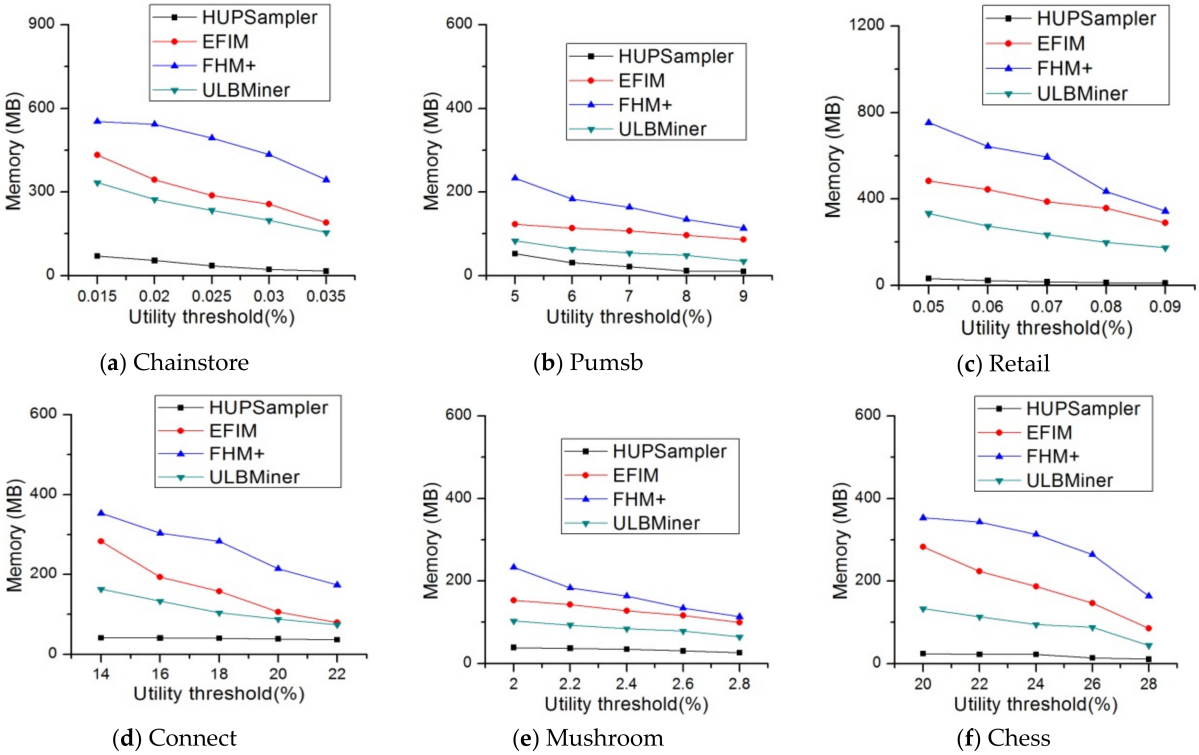


Figure 8. Memory usage comparisons by four algorithms.

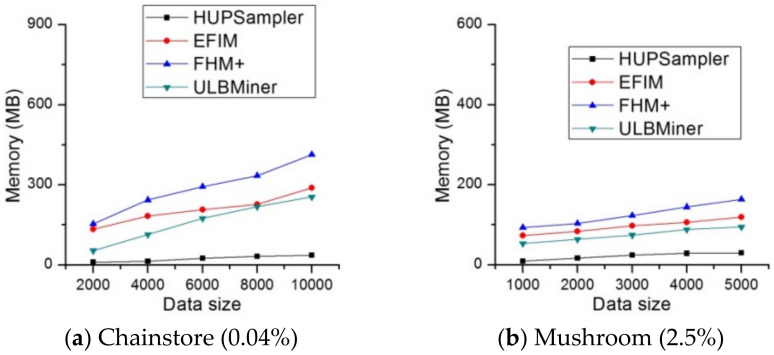


Figure 9. Memory usage under different data sizes.

4.3. Utility Distribution

Figure 10 returns the distribution of utility values for 100 runs under different minimum utility parameters. The utility value of HUPSampler is well controlled. For example, on Retail, the extraction times with utility greater than 1500 account for 92.5%, and on Connect, the extraction times with utility greater than 300 account for 90.5%. This result shows that our sampling method can relatively return patterns with high utility with high time-space efficiency.

In addition, Table 9 shows the ratio of the HUPs comparison of HUPSampler to the state-of-the-art algorithm ULBMiner, which returns all exact HUPs under length constraints with utility thresholds of 0.025%, 7%, 0.07%, 18%, 2.4%, and 24%, respectively. Thanks to our extraction mode, it can be seen that HUPSampler can return most HUPs under the length constraints. For example, on Chainstore, HUPSampler returns more than 90% of HUPs; the diversity of the return patterns could be guaranteed by the high probability with low time-space computations.

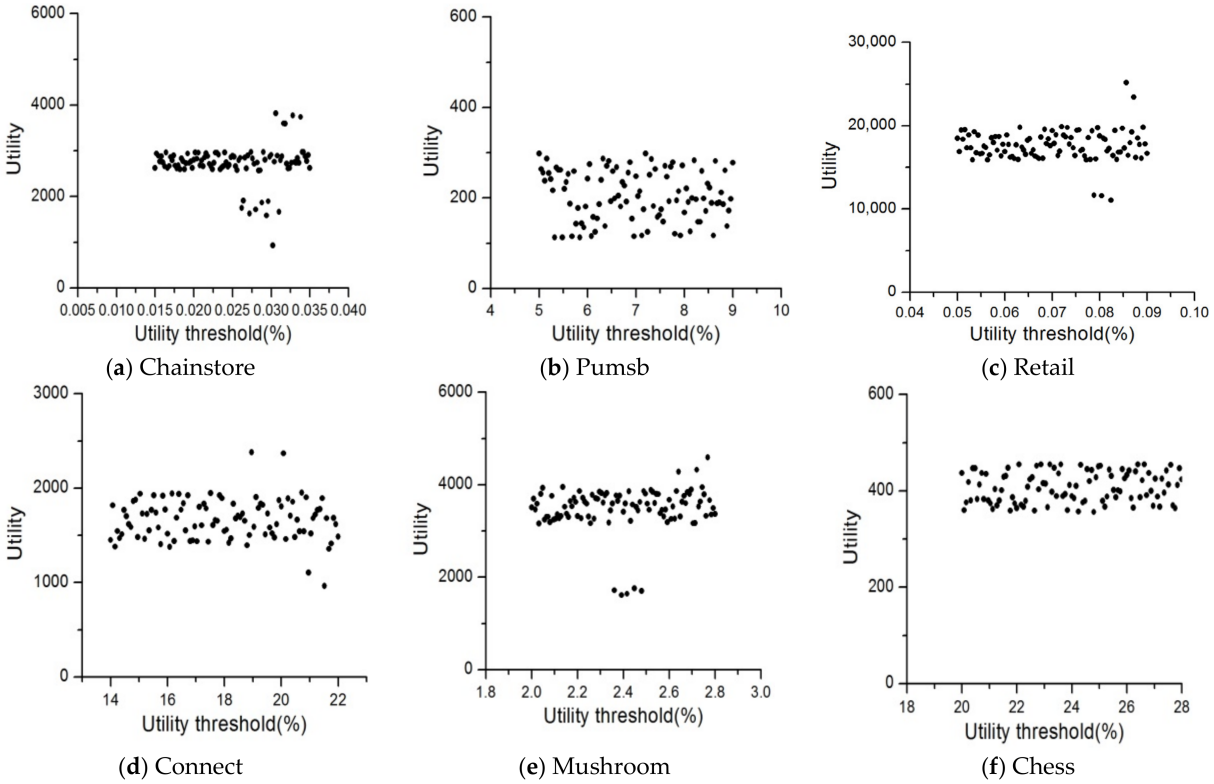


Figure 10. Utility distribution by HUPSampler.

Table 9. Ratio of HUPs by running HUPSampler 1000 times.

Dataset	Length Constraints					
	(10, 20)	(10, 30)	(10, 40)	(10, 50)	(10, 60)	(10, 70)
Chainstore	0.913	0.933	0.934	0.954	0.964	0.966
Pumsb	0.892	0.912	0.923	0.925	0.931	0.931
Retail	0.873	0.895	0.904	0.905	0.912	0.921
Connect	0.852	0.874	0.883	0.883	0.893	0.902
Mushroom	0.812	0.823	0.835	0.836	0.846	0.875
Chess	0.823	0.832	0.843	0.851	0.861	0.863

4.4. Case Study

COVID-19 has seriously affected people's lives [34]. The spread of the virus will be accelerated in the crowd gathering environment. Once there are new cases, the best way is to conduct epidemiological investigation and study the path trajectories of patients for finding people associated with these locations who may be potential infectors. The path information of patients is very important. Accurate identification of these key places is of great significance to identify the high-risk areas with epidemic development, so as to make corresponding prevention measures to control the spread of the epidemic.

We use the data obtained from http://wsjkw.hebei.gov.cn/html/yqtb/index_30.jhtml (accessed on 10 December 2022). Such data is textual and contains the path trajectories and location information of COVID-19 patients. The span time was from 3 January 2021 to 28 January 2021. A total of 698 cases were collected, involving 1361 locations. A path example of an infected individual could be known as "Xiao Guo Zhuang, Shi Jia Zhuang 5th Hospital, Xin Le Hospital, Zhang Jia Zhuang"; it can be identified as a transaction in Table 1. Here, the internal numbers and profits (importance) of the places cannot be known like the data in Tables 1 and 2 can. Since we cannot give the number and profits of locations

subjectively, the utility setting method in the literature [32,33] is adopted. The number of places in the transactions (internal utility value) is a random integer less than 10 and greater than 0; the external utility value (profits) of the item is also a randomly generated value (greater than or equal to 0.01, less than or equal to 10.0). We use HUPSampler to get the path with high utility. Table 10 shows the length and utility distribution by run the algorithm 100 times when the minimum utility thresholds are 0.25, 0.5, and 0.7, respectively. According to the 300 sampling results based on different utility thresholds, Table 11 shows the top 10 single locations sorted by the average high utility (sum of utility/sampling times). Such places may be high-risk and should be paid more attention. Some epidemic prevention measures may be taken firstly in such places.

Table 10. Length proportion and utility distribution.

Minimum Utility Threshold	Proportion (k < 3)	Proportion ($3 \leq k \leq 5$)	Proportion (k > 5)	Average Utility
0.25	23%	61%	16%	34.6
0.5	35%	57%	8%	134.4
0.75	12%	86%	2%	125.6

Table 11. Top 10 location combinations with high utility.

Top 10 Places with High Utility	Average Utility
Xiao Guo Zhuang	212.6
Liu Jia Zuo	178.5
Shi Jia Zhuang 5th Hospital	126.4
Xin Le Hospital	87.6
Xiao Guo Zhuang Primary School	75.8
Gao Cheng Hospital	70.9
No. 7 Middle School	67.5
Hao Yun Lai Restaurant	60.8
Zeng Cun	58.6
Ou Jing Yuan	50.6

5. Conclusions and Feature Work

We introduce an efficient algorithm named HUPSampler to return HUP randomly with high time-space efficiency. Such an algorithm could be divided into two parts. First, it extracts an integer k proportionally to the utility based on a random length interval determined by the users, and then it outputs a HUP of length k randomly. For calculating one HUP of k in a short time, our approach uses a pattern growth method to output the result in a timely manner based on a random tree structure. The experimental results on real datasets demonstrate the feasibility and advantages of the method. A case study of COVID-19 is also introduced to show the effectiveness and applicability of our approach. There are several possible improvements, which could be done in the future. For example, HUPSampler is designed for returning one HUP, so the randomly selected HUP may not have utility representativeness when the transaction is homologous. In addition, if the transaction is uncertain, that is, the transaction itself has a probability, the utility proportion calculation method of the extracted pattern should also be different. We will investigate these problems in our future work.

Author Contributions: Formal analysis, H.T. and L.W.; Methodology, H.T. and J.W.; Writing—original draft, H.T. and L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Social Science Foundation of China grant number 21BGL088.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cheng, J.; Xie, Y.; Zhang, J. Industry structure optimization via the complex network of industry space: A case study of Jiangxi Province in China. *J. Clean. Prod.* **2022**, *338*, 130602. [CrossRef]
2. Wang, E.T.; Chen, A.L. Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis. *Data Min. Knowl. Discov.* **2011**, *23*, 252–299. [CrossRef]
3. Cheng, J.; Luo, X.W. Analyzing the land leasing behavior of the government of Beijing, China, via the multinomial logit model. *Land* **2022**, *11*, 376. [CrossRef]
4. Tseng, V.; Shie, B.; Wu, C.; Yu, P. Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. *IEEE Trans. Knowl. Data Eng.* **2013**, *25*, 1772–1786. [CrossRef]
5. Nguyen, H.; Le, N.; Bui, H.; Le, T. A new approach for efficiently mining frequent weighted utility patterns. *Appl. Intell.* **2022**, *53*, 121–140. [CrossRef]
6. Tung, N.T.; Nguyen, L.T.; Nguyen, T.D.; Fourier-Viger, P.; Nguyen, N.T.; Vo, B. Efficient mining of cross-level high-utility itemsets in taxonomy quantitative databases. *Inf. Sci.* **2022**, *587*, 41–62. [CrossRef]
7. Duong, Q.H.; Fournier-Viger, P.; Ramampiaro, H.; Norvag, K.; Dam, T.L. Efficient high utility itemset mining using buffered utility-lists. *Appl. Intell.* **2018**, *48*, 1859–1877. [CrossRef]
8. Fournier-Viger, P.; Wu, C.W.; Souleymane, Z.; Vincent, S. FHM: Faster high-utility itemset mining using estimated utility co-occurrence pruning. In *Foundations of Intelligent Systems*, 1st ed.; Springer: Cham, Switzerland, 2014; pp. 83–92.
9. Liu, J.; Wang, K.; Fung, B. Direct Discovery of High Utility Itemsets without Candidate Generation. In Proceedings of the 2012 IEEE 12th International Conference on Data Mining (ICDM), Brussels, Belgium, 10–13 December 2012; pp. 984–989.
10. Souleymane, Z.; Fournier-Viger, P.; Jerry, C.; Lin, W.; Wu, C.W.; Vincent, S.T. EFIM: A fast and memory efficient algorithm for high-utility itemset mining. *Knowl. Inf. Syst.* **2017**, *51*, 595–625.
11. Liu, M.; Qu, J. Mining high utility itemsets without candidate generation. In Proceedings of the 21st ACM International Conference on Information and Knowledge Management, Maui, HI, USA, 29 October–2 November 2012; pp. 55–64.
12. Fournier-Viger, P.; Lin, J.C.; Dong, Q.; Dam, T. FHM+: Faster high-utility itemset mining using length upper-bound reduction. In Proceedings of the International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, Berlin/Heidelberg, Germany, 2–4 August 2016; pp. 115–127.
13. Jenkins, S.; Walzer-Goldfeld, S.; Riondato, M. SPEck: Mining Statistically-significant Sequential Patterns Efficiently with Exact Sampling. *Data Mining Knowl. Disc.* **2022**, *36*, 1575–1599. [CrossRef]
14. Pellegrina, L.; Cousins, C.; Vandin, F.; Riondato, M. McRapper: Monte-Carlo Rademacher Averages for Poset Families and Approximate Pattern Mining. *ACM Trans. Knowl. Discov. Data* **2022**, *16*, 124. [CrossRef]
15. Djenouri, Y.; Comuzzi, M. Combining Apriori heuristic and bio-inspired algorithms for solving the frequent itemsets minin problem. *Inf. Sci.* **2017**, *420*, 1–15. [CrossRef]
16. Pietracaprina, A.; Riondato, M.; Upfal, E.; Vandin, F. Mining top-k frequent itemsets through progressive sampling. *Data Min. Knowl. Disc.* **2010**, *21*, 310–326. [CrossRef]
17. Lin, J.C.W.; Zhang, Y.; Zhang, B.; Fournier-Viger, P.; Djenouri, Y. Hiding sensitive itemsets with multiple objective optimization. *Soft Comput.* **2019**, *23*, 12779–12797. [CrossRef]
18. Tseng, V.S.; Wu, C.W.; Fournier-Viger, P.; Yu, P.S. Efficient Algorithms for Mining Top-K High Utility Itemsets. *IEEE Trans. Knowl. Data Eng.* **2016**, *28*, 54–67. [CrossRef]
19. Yun, U.; Ryang, H.; Ryu, K.H. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. *Expert Syst. Appl.* **2014**, *41*, 3861–3878. [CrossRef]
20. Zhang, C.; Zhang, S.; Webb, G.I. Identifying approximate itemsets of interest in large databases. *Appl. Intell.* **2003**, *18*, 91–104. [CrossRef]
21. Gan, W.; Lin, J.C.W.; Zhang, J. Fast utility mining on sequence data. *IEEE Trans. Cybern.* **2021**, *51*, 487–500. [CrossRef]
22. Bashir, S.; Lai, D. Mining Approximate Frequent Itemsets Using Pattern Growth Approach. *Inf. Technol. Control* **2022**, *50*, 627–644. [CrossRef]
23. Yan, C.; Aijun, A. Approximate Parallel High Utility Itemset Mining. *Big Data Res.* **2016**, *6*, 26–42.
24. Diego, S.; Leonardo, P.; Matteo, C.; Fabio, V. SPRISS: Approximating Frequent K-mers by Sampling Reads, and Applications. *Bioinformatics* **2022**, *38*, 3343–3350.
25. Cheng, J. Analysis of the factors influencing industrial land leasing in Beijing of China based on the district-level data. *Land Use Policy* **2022**, *122*, 106389. [CrossRef]
26. Han, Y.; Moghaddam, M. Analysis of sentiment expressions for user-centered design. *Expert Syst. Appl.* **2021**, *171*, 114604. [CrossRef]
27. Yin, P.; Cheng, J. A MySQL-based software system of urban land planning database of Shanghai in China. *CMES-Comp. Model Eng.* **2023**, *135*, 2387–2405. [CrossRef]

28. Fournier-Viger, P.; Gomariz, A.; Gueniche, T. Spmf: A java open source pattern mining library. *J. Mach. Learn. Res.* **2014**, *15*, 3389–3393.
29. Diop, A.; Giacometti, D.; Li, A.S. Sequential Pattern Sampling with Norm Constraints. In Proceedings of the IEEE International Conference on Data Mining (ICDM), Singapore, 17–20 November 2018; pp. 89–98.
30. Diop, L. High Average-Utility Itemset Sampling Under Length Constraints. In Proceedings of the 26th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Berlin/Heidelberg, Germany, 16–19 May 2022; pp. 134–148.
31. Wang, L. An Algorithm for Mining Fixed-Length High Utility Itemsets. In *Lecture Notes in Computer Science*, 1st ed.; Springer: Cham, Switzerland, 2022; pp. 3–20.
32. Ahmed, C.F.; Tanbeer, S.K.; Jeong, B.S.; Lee, Y.K. Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases. *IEEE Trans. Knowl. Data Eng.* **2009**, *21*, 1708–1721. [CrossRef]
33. Li, Y.C.; Yeh, J.S.; Chang, C.C. Isolated items discarding strategy for discovering high utility itemsets. *Data Knowl. Eng.* **2008**, *64*, 198–217. [CrossRef]
34. Cheng, J.; Yin, P. Analysis of the complex network of the urban function under the lockdown of COVID-19: Evidence from Shenzhen in China. *Mathematics* **2022**, *10*, 2412. [CrossRef]